

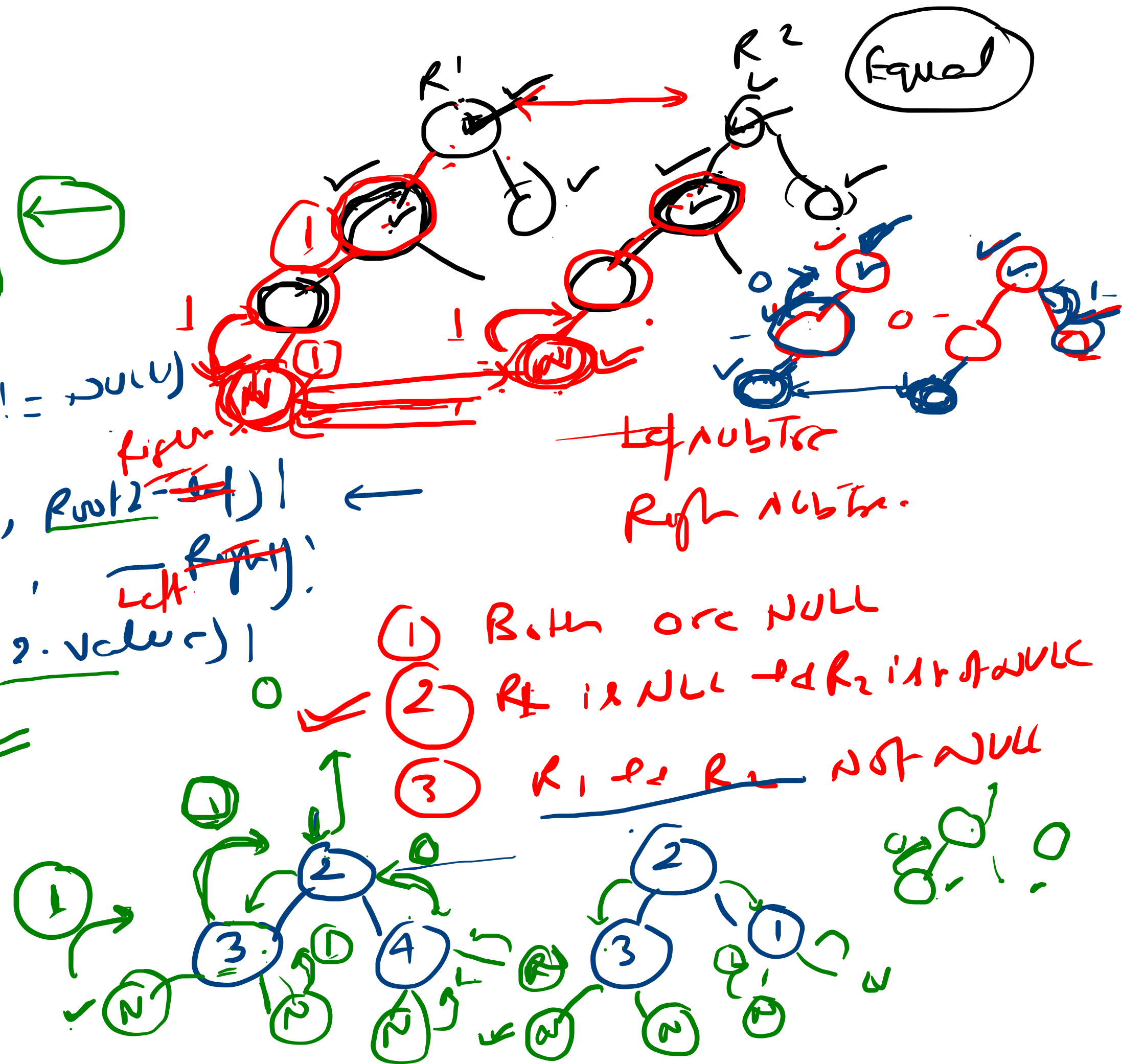
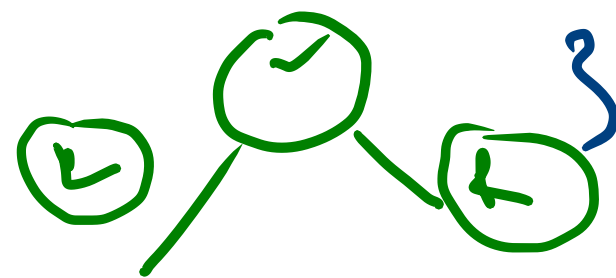
int check Tree(Node* Root1 , Node* Root2)

if (Root1 == Root2 == NULL) \leftarrow
return(1)!

~~else if~~ (Root1 != NULL && Root2 != NULL)
int LV = checkTree (Root1 -> Left, Root2 -> Left)
int RV = checkTree (Root1 -> Right, Root2 -> Right)

Bottom X = (Root1->value == Root2->value)
return (LV && RV && X) \leftarrow


else return(0)!



① Convert a Binary Tree to Mirror Image

```
void Mirror(Node* Root)
```

```
{
```

```
    if (Root == NULL) 
```

```
        return;
```

```
    // ne
```

```
    Mirror(Node->Left);
```

```
    Mirror(Node->Right);
```

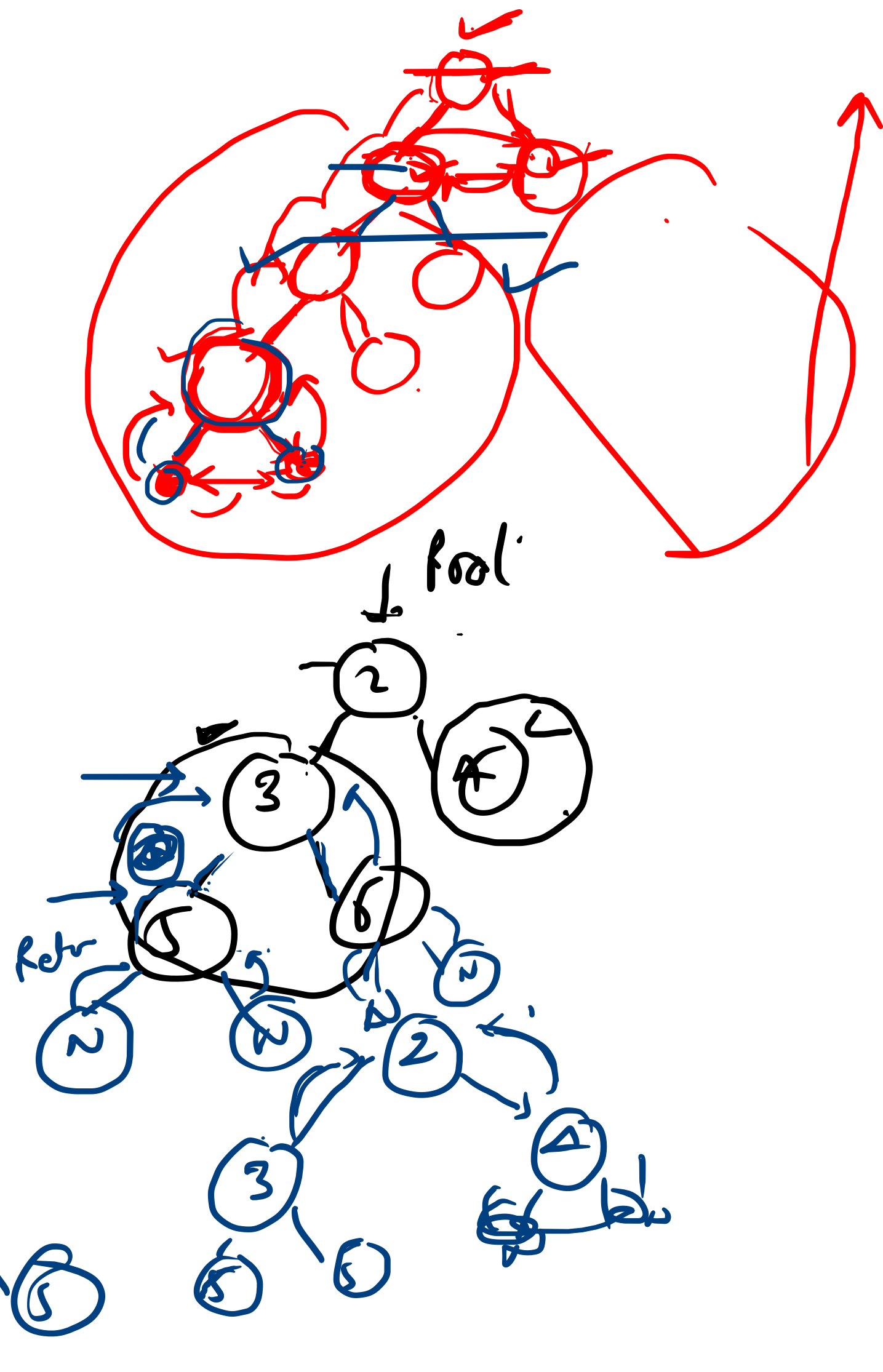
```
    Node* Temp = Node->Left;
```

```
    Node->Left =
```

```
    Node->Right = Temp;
```

```
    return;
```

```
}
```



① Binary Tree Labd with posibility

② height - print all items

③ Print BT

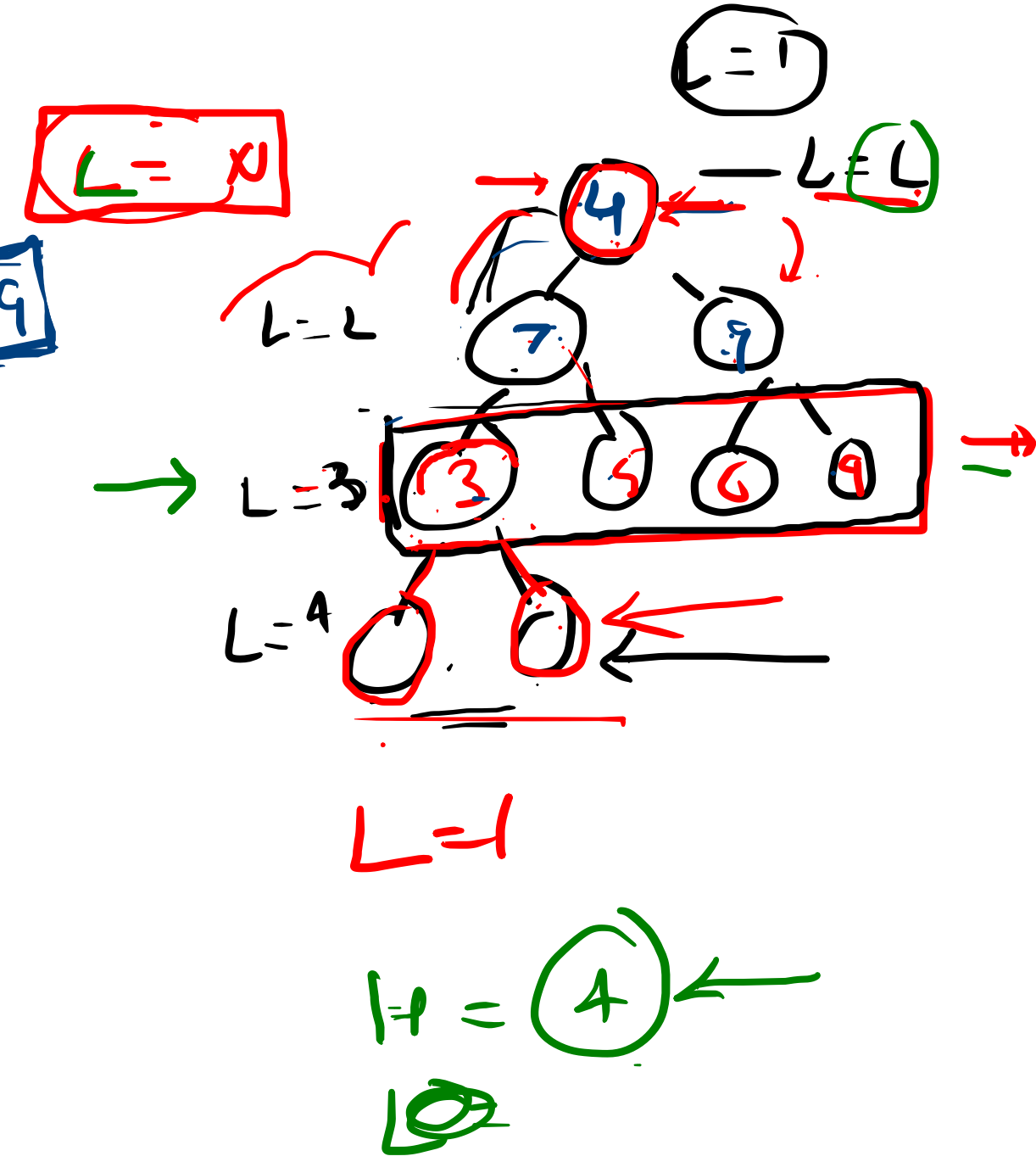
④ height = height-BT (Root);

⑤ for (i = 1, i ≤ height; i++)
PrintBT (Root, i);

⑥ void PrintBT (Node* Root, int level)
{
if (Root == null) return;
else if (level == 1) Print (Root->item);
else
PrintBT (Root->Left, level-1);
PrintBT (Root->Right, level-1);
}

Level == null

2 3 5 6 9



④ 7 9
return

① Root To Leaf Path
 int x[1000]
 print BT (Root, x, 0)

void print BT (Node* Root, int x[], int index)

{
 if Root == NULL
 return

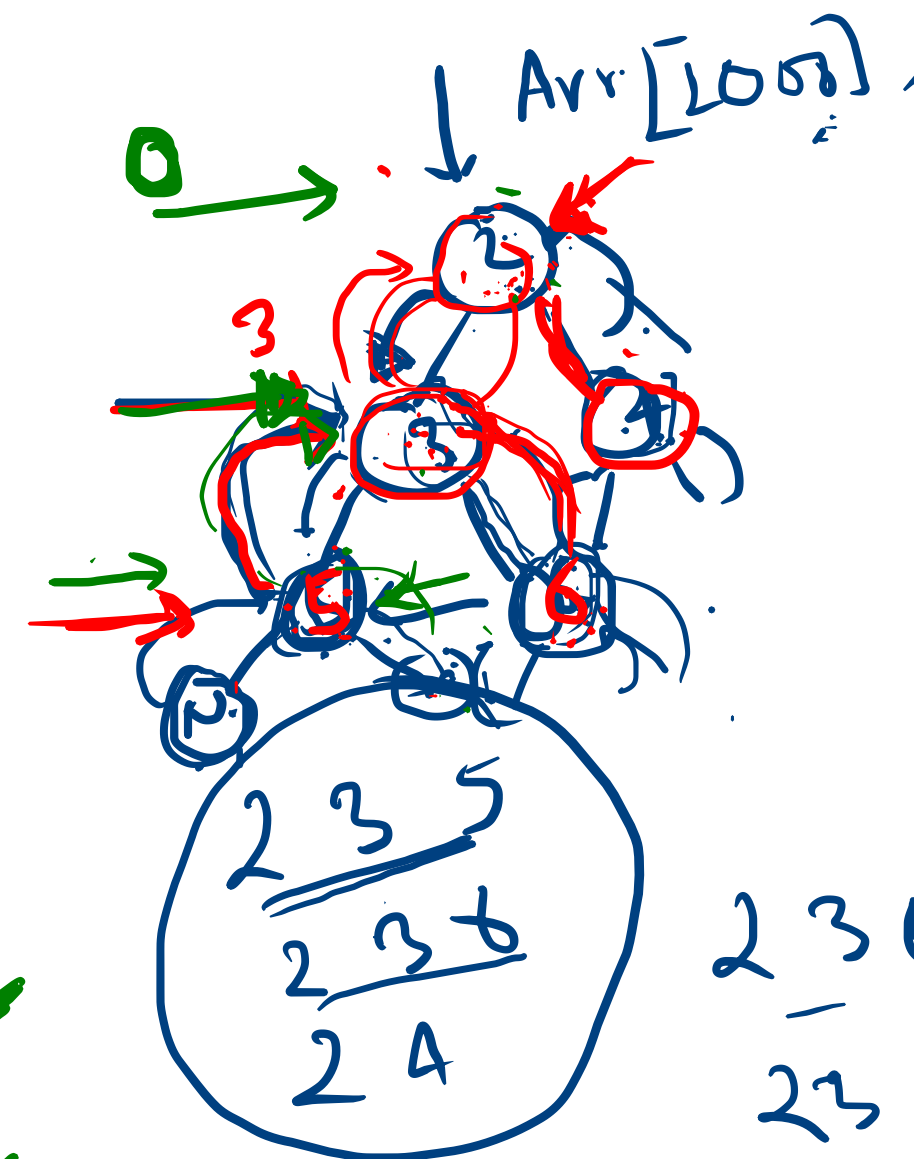
int x[index] = Root->val;

print BT (Root->Left, x, index+1);
 print BT (Root->Right, x, index+1);

if (Root->Left == NULL || Root->Right == NULL)

print (x, 0, index)

index = index - 1;



2 3 5
 2 3 6
 2 4
 Add item
 Post order

Add item
 LL
 RL
 Pre order
 In order
 Post order

Binary Search Tree:

- ① $L < M$ ✓
- ② $R > M$ ✓

③ FST, RST → BST ✓

Node insertion BST (Node * Root, int item)

```

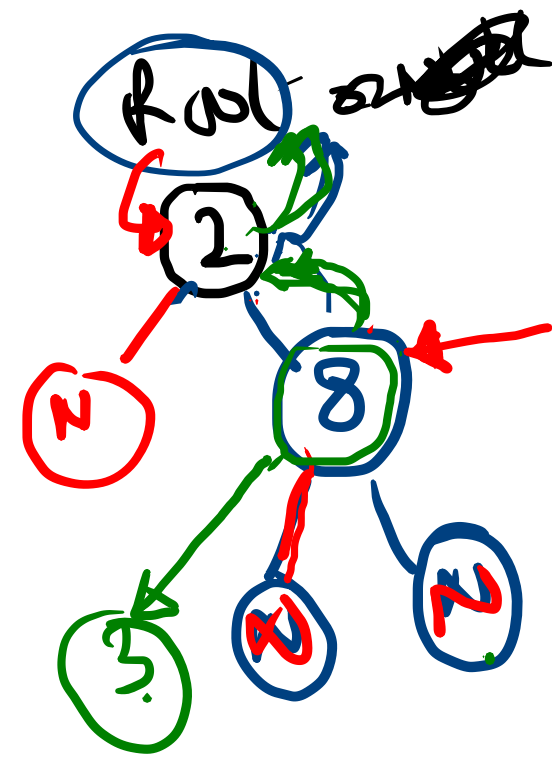
{
    if (Root == NULL) {
        Temp Node * = creat Node();
        Root = Temp;
        return (Root);
    }

```

```

    if (item < Root->value) {
        Root->left = insertion BST (Root->left, item);
    }
    else if (item > Root->value) {
        Root->right = insertion BST (Root->right, item);
    }
    return (Root);
}

```



```

int main()
{
    Node * Root = NULL;
    insert BST (Root, 2);
    insert BST (Root, 8);
    insert BST (Root, 3);
}

```

Assignment 8:

- 2

finding or elements Node Addr
 null
max and min element

